DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY AND RESEARCH

CE261 – Data Structures and Algorithms

Subject Coordinator:

Prof. Khushi Patel, Assistant Professor

Subject Faculties:

Prof. Shraddha Vyas, Prof. Khushi Patel

Devang Patel Institute of Advance Technology And Research
Charotar University of Science and Technology

Objectives

- Understand and Implement Algorithms and core Data Structures such as stack, queue, hash table, priority queue, binary search tree and graph in programming language.
- Analyze data structures in storage, retrieval and computation of ordered or unordered data
- Compare alternative implementations of data structures with respect to demand and performance
- Describe and evaluate the properties, operations, applications, strengths and weaknesses of different data structures.
- Apply and select the most suitable data structures to solve programming challenges.
- Discover advantages and disadvantages of specific algorithms

Introduction to Data structure

- Data structures are a way of organizing and storing data in a computer so that it can be accessed and manipulated efficiently. They provide a systematic and organized approach to represent, store, and manage data elements and their relationships.
- Data structures define how data is arranged, how it can be accessed, and what operations can be performed on it. They are a fundamental concept in computer science and are essential for building efficient algorithms and solving various computational problems.
- Data structures can be classified into two main types:
    1. Primitive Data Structures
    2. Non - Primitive Data Structures

Introduction to Data structure

- **Primitive Data Structures:** These are basic data types that are built into a programming language and are generally considered the most basic data types.
- They are called primitive because they are not composed of any other data types. Examples of primitive data types include integers, float, characters, and boolean values. They represent single values and have fixed memory requirements.

- **Non - Primitive Data Structures:** These are more complex structures that are built using primitive data types and provide a higher level of abstraction.
- They allow data to be organized in a more sophisticated manner.
- Some common abstract data structures include arrays, linked lists, stacks, queues, trees, graphs, and hash tables.

Need of Data Structure

- As applications are becoming more complex and the amount of data is increasing every day, which may lead to problems with data searching, processing speed, multiple requests handling, and many more.
- Data Structures support different methods to organize, manage, and store data efficiently.
- With the help of Data Structures, we can easily traverse the data items. Data Structures provide Efficiency, Reusability, and Abstraction.
- It plays an important role in enhancing the performance of a program because the main function of the program is to store and retrieve the user's data as fast as possible.
- Learning data structure and algorithms will help to become a better programmer.

Objective of Data Structures

- Data Structures satisfy two objectives:
- **Correctness:** Data Structures are designed to operate correctly for all kinds of inputs based on the domain of interest. In order words, correctness forms the primary objective of Data Structure, which always depends upon the problems that the Data Structure is meant to solve.
- **Efficiency:** Data Structures also requires to be efficient. It should process the data quickly without utilizing many computer resources like memory space.
- In a real-time state, the efficiency of a data structure is a key factor in determining the success and failure of the process.

Introduction to Data structure

- Data structures and algorithms are closely related and interconnected concepts in computer science.
- Relationship between data structures and algorithms
- **Efficient Data Representation:** Data structures provide a way to organize and store data in memory effectively. The choice of data structure affects the performance of algorithms.
- For example, using an array for random access or a linked list for dynamic insertion/deletion can significantly impact the time complexity of algorithms that work on these data structures.
- **Algorithm Design:** Algorithms are step-by-step procedures or instructions to solve specific problems. The design of an algorithm often depends on the underlying data structure used for the problem. Different data structures may require different algorithms to achieve the desired results efficiently.

Introduction to Data structure

- **Searching and Sorting**: Data structures play an important role in searching and sorting algorithms.
- For example, arrays are suitable for binary search, which requires random access, while binary search trees enable efficient searching through their hierarchical structure.
- **Insertion and Deletion:** The choice of data structure affects the efficiency of insertion and deletion operations.
- For instance, linked lists at constant-time insertions and deletions at the beginning or end, while arrays might be slower due to the need for shifting elements.
- **Space and Time Complexity:** Data structures influence the time and space complexity of algorithms. Using the right data structure can lead to more efficient algorithms with lower time complexity and reduced memory usage.

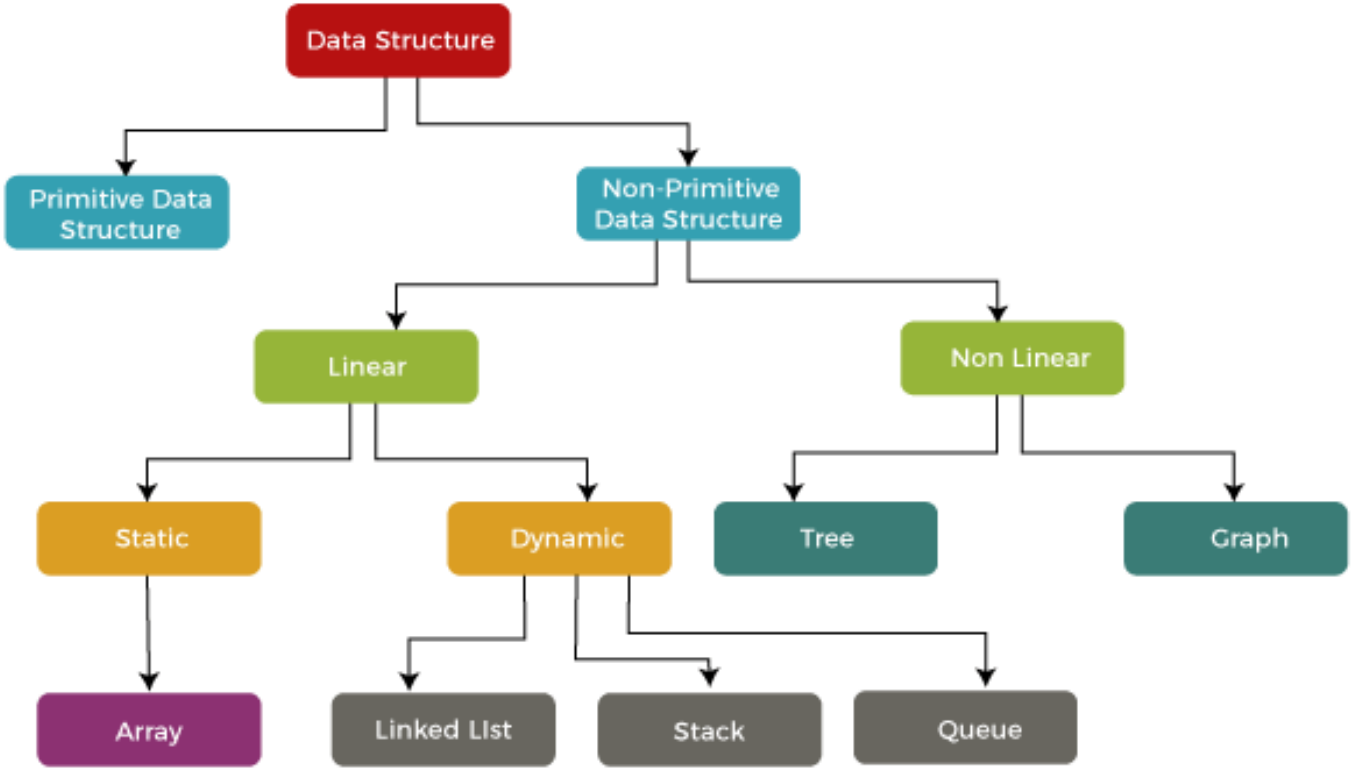
Introduction to Data structure

- **Dynamic Memory Management:** Some data structures, like linked lists, dynamically allocate memory during runtime, which allows for flexible data management.
- Algorithms designed to work with these data structures must handle memory management accordingly.
- **Recursive Algorithms:** Recursive algorithms, which call themselves to solve a problem, often rely on data structures like stacks to manage function calls and maintain the execution context.
- **Problem Solving:** When approaching a computational problem, choosing an appropriate data structure is often one of the first steps. Different data structures provide different strengths, and understanding their properties helps in devising efficient algorithms.

Introduction to Data structure

- Data structures provide the foundation for storing and organizing data, while algorithms define the steps to manipulate and process that data to solve specific problems.
- The selection and design of data structures and algorithms have a significant impact on the efficiency and performance of software applications.
- Choosing the right data structure for a given algorithm and problem is essential for achieving optimal results in terms of time and space complexity.

Type of Data Structures



Type of Data Structures

- **Linear Data Structures**
- A data structure that preserves a linear connection among its data elements is known as a Linear Data Structure.
- The arrangement of the data is done linearly, where each element consists of the successors and predecessors except the first and the last data element.
- Based on memory allocation, the Linear Data Structures are further classified into two types:
- **Static Data Structures**
- **Dynamic Data Structures**

Type of Data Structures

- **Static Data Structures:** The data structures having a fixed size are known as Static Data Structures.
- The memory for these data structures is allocated at the compiler time, and their size cannot be changed by the user after being compiled; however, the data stored in them can be altered.
- The Array is the best example of the Static Data Structure as they have a fixed size, and its data can be modified later.
- **Dynamic Data Structures:** The data structures having a dynamic size are known as Dynamic Data Structures.
- The memory of these data structures is allocated at the run time, and their size varies during the run time of the code. Moreover, the user can change the size as well as the data elements stored in these data structures at the run time of the code.
- Linked Lists, Stacks, and Queues are common examples of dynamic data structures

Type of Data Structures

- **Non-Linear Data Structures**
- Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can’t traverse all the elements in a single run only.
- Examples of non-linear data structures are trees and graphs.

Types of Data Structures

- **Array:** An array is a linear data structure and it is a collection of items stored at contiguous memory locations.
- The idea is to store multiple items of the same type together in one place.
- It allows the processing of a large amount of data in a relatively short period. The first element of the array is indexed by 0.
- There are different operations possible in an array, like Searching, Sorting, Inserting, Traversing, Reversing, and Deleting.

Types of Data Structures

- **Characteristics of an Array:**
- Arrays use an index-based data structure which helps to identify each of the elements in an array easily using the index.
- If a user wants to store multiple values of the same data type, then the array can be utilized efficiently.
- An array can also handle complex data structures by storing data in a two-dimensional array.
- An array is also used to implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.
- The search process in an array can be done very easily.

Types of Data Structures

- **Operations performed on array:**
- **Initialization:** An array can be initialized with values at the time of declaration or later using an assignment statement.
- **Accessing elements**: Elements in an array can be accessed by their index, which starts from 0 and goes up to the size of the array minus one.
- **Searching for elements:** Arrays can be searched for a specific element using linear search or binary search algorithms.
- **Sorting elements:** Elements in an array can be sorted in ascending or descending order using algorithms like bubble sort, insertion sort, or quick sort.
- **Inserting elements**: Elements can be inserted into an array at a specific location, but this operation can be time-consuming because it requires shifting existing elements in the array.

Types of Data Structures

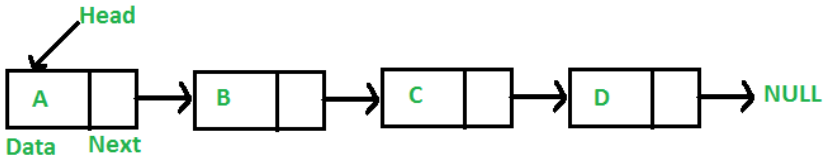
- **Deleting elements:** Elements can be deleted from an array by shifting the elements that come after it to fill the gap.
- **Updating elements:** Elements in an array can be updated or modified by assigning a new value to a specific index.
- **Traversing elements:** The elements in an array can be traversed in order, visiting each element once.

Types of Data Structures

- **Applications:**
- It helps in implementing a sorting algorithm.
- It is used to implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.
- In games like online chess, where the player can store his past moves as well as current moves. It indicates a hint of position.
- Image Processing
- Mathematical Operations: They are used in linear algebra operations such as matrix multiplication, vector operations, and solving equations.

Types of Data Structures

- **Linked list:**
- A linked list is a linear data structure that is used to maintain a list-like structure in the computer memory. It is a group of nodes that are not stored at contiguous locations. Each node of the list is linked to its adjacent node with the help of pointers.



- **Types of linked lists:**
- Singly-linked list
- Doubly linked list
- Circular linked list
- Doubly circular linked list

Types of Data Structures

- **Characteristics of a Linked list:**
- A linked list uses extra memory to store links.
- During the initialization of the linked list, there is no need to know the size of the elements.
- The first node of the linked list is called the Head.
- The next pointer of the last node always points to NULL.
- In a linked list, insertion and deletion are possible easily.
- Each node of the linked list consists of a pointer/link which is the address of the next node.
- Linked lists can shrink or grow at any point in time easily.

Types of Data Structures

- **Operations performed on Linked list:**
- A linked list is a linear data structure where each node contains a value and a reference to the next node. Here are some common operations performed on linked lists:
- **Initialization:** A linked list can be initialized by creating a head node with a reference to the first node. Each subsequent node contains a value and a reference to the next node.
- **Inserting elements:** Elements can be inserted at the head, tail, or at a specific position in the linked list.
- **Deleting elements:** Elements can be deleted from the linked list by updating the reference of the previous node to point to the next node, effectively removing the current node from the list.
- **Searching for elements:** Linked lists can be searched for a specific element by starting from the head node and following the references to the next nodes until the desired element is found.

Types of Data Structures

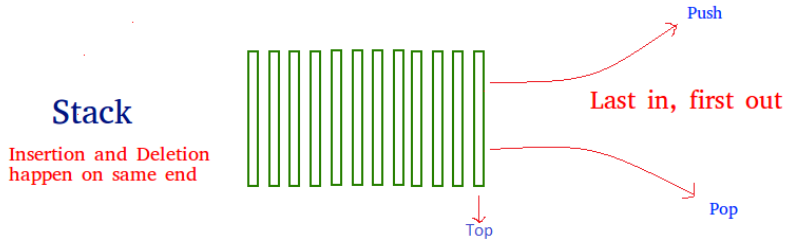
- **Updating elements:** Elements in a linked list can be updated by modifying the value of a specific node.
- **Traversing elements:** The elements in a linked list can be traversed by starting from the head node and following the references to the next nodes until the end of the list is reached.
- **Reversing a linked list:** The linked list can be reversed by updating the references of each node so that they point to the previous node instead of the next node.

Types of Data Structures

- **Real-Life Applications of a Linked list:**
- A linked list is used in Round-Robin scheduling to keep track of the turn in multiplayer games.
- It is used in image viewer. The previous and next images are linked, and hence can be accessed by the previous and next buttons.
- In a music playlist, songs are linked to the previous and next songs.
- They are used to store the history of the visited page.
- They are used to perform undo operations.
- Linked lists are used to display social media feeds.
- Task Management

Types of Data Structures

- **Stack:**
- Stack is a linear data structure that follows a particular order in which the operations are performed.
- The order is LIFO(Last in first out). Entering and retrieving data is possible from only one end.
- The entering and retrieving of data is also called push and pop operation in a stack. There are different operations possible in a stack like reversing a stack using recursion, Sorting, Deleting the middle element of a stack, etc.



Types of Data Structures

- **Characteristics of a Stack:**
- Stack has various different characteristics which are as follows:
- Stack is used in many different algorithms like Tower of Hanoi, tree traversal, recursion, etc.
- Stack is implemented through an array or linked list.
- It follows the Last In First Out operation i.e., an element that is inserted first will pop in last and vice versa.
- The insertion and deletion are performed at one end i.e. from the top of the stack.
- In stack, if the allocated space for the stack is full, and still anyone attempts to add more elements, it will lead to stack overflow.

Types of Data Structures

- **Operation performed on stack :**
- A stack is a linear data structure that implements the Last-In-First-Out (LIFO) principle. Here are some common operations performed on stacks:
- **Push:** Elements can be pushed onto the top of the stack, adding a new element to the top of the stack.
- **Pop:** The top element can be removed from the stack by performing a pop operation, effectively removing the last element that was pushed onto the stack.
- **Peek:** The top element can be inspected without removing it from the stack using a peek operation.
- **IsEmpty:** A check can be made to determine if the stack is empty.
- **Size:** The number of elements in the stack can be determined using a size operation.

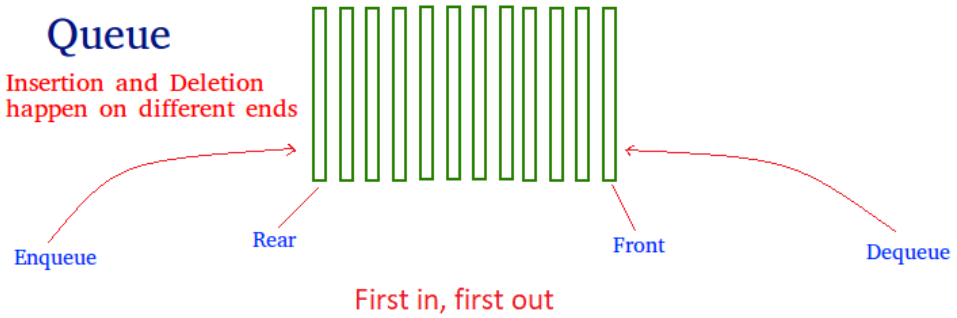
Types of Data Structures

- **Applications**
- Function Calls
- Expression Evaluation
- Undo/Redo Functionality
- Memory Management
- Imagine a real-life analogy of a stack as a stack of plates. You can only add (push) or remove (pop) plates from the top of the stack. The last plate you put on the stack is the first one to be removed when you pop an element.

Types of Data Structures

Queue:

- Queue is a linear data structure that follows a particular order in which the operations are performed. The order is First In First Out(FIFO) i.e. the data item stored first will be accessed first.
- In this, entering and retrieving data is not done from only one end.
- An example of a queue is any queue of consumers for a resource where the consumer that came first is served first.



Types of Data Structures

Characteristics of a Queue:

- The queue is a FIFO (First In First Out) structure.
- To remove the last element of the Queue, all the elements inserted before the new element in the queue must be removed.
- A queue is an ordered list of elements of similar data types.

Types of Data Structures

- **Operation performed on queue:**
- A queue is a linear data structure that implements the First-In-First-Out (FIFO) principle. Here are some common operations performed on queues:
- **Enqueue:** Elements can be added to the back of the queue, adding a new element to the end of the queue.
- **Dequeue:** The front element can be removed from the queue by performing a dequeue operation, effectively removing the first element that was added to the queue.
- **Peek:** The front element can be inspected without removing it from the queue using a peek operation.
- **IsEmpty:** A check can be made to determine if the queue is empty.
- **Size:** The number of elements in the queue can be determined using a size operation.

Types of Data Structures

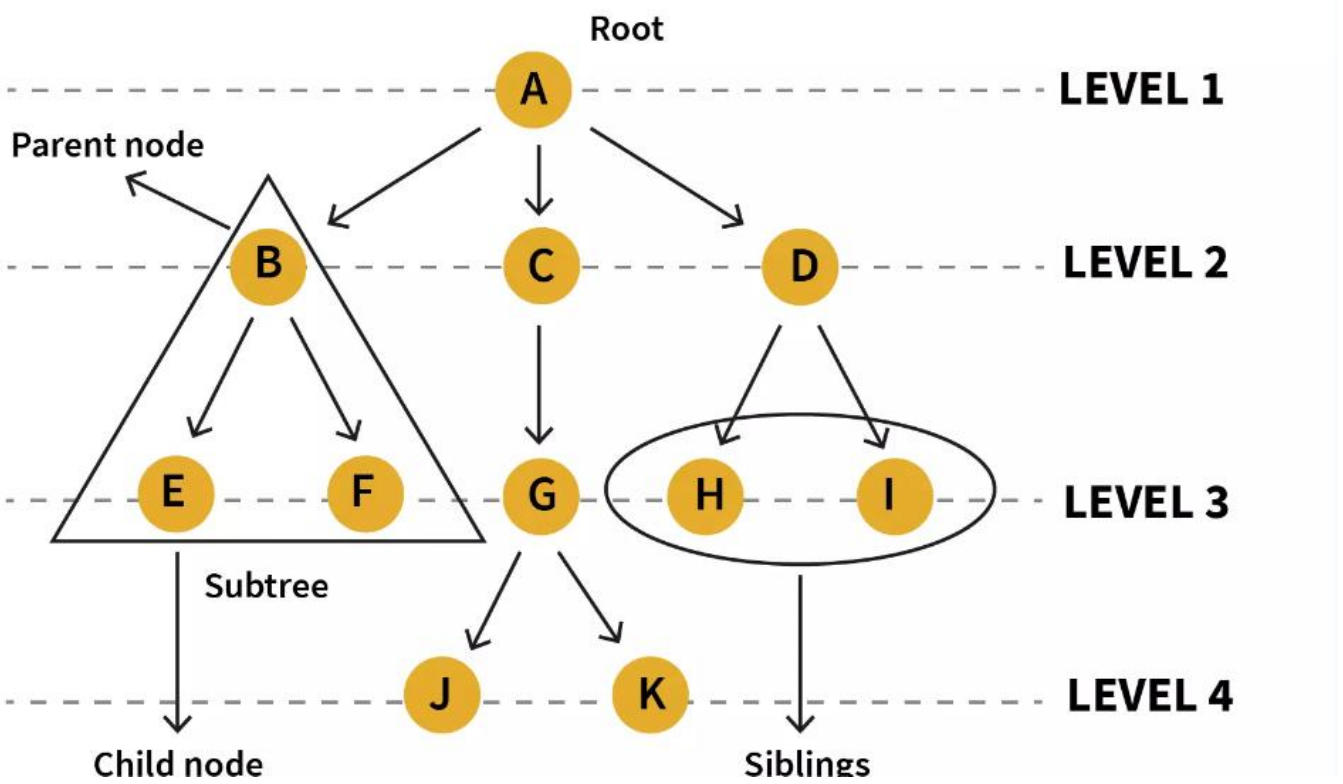
- **Real-Life Applications of Queue:**
- A real-world example of a queue is a single-lane one-way road, where the vehicle that enters first will exit first.
- A more real-world example can be seen in the queue at the ticket windows.
- Print
- Task Scheduling
- Message Queue
- Resource Managemen

Types of Data Structures

- **Tree:**
- A tree is a non-linear and hierarchical data structure where the elements are arranged in a tree-like structure.
- In a tree, the topmost node is called the root node. Each node contains some data, and data can be of any type.
- It consists of a central node, structural nodes, and sub-nodes which are connected via edges. Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure. A tree has various terminologies like Node, Root, Edge, Height of a tree, Degree of a tree, etc.
- **There are different types of Tree-like**
- Binary Tree,
- AVL Tree,
- B-Tree, etc.

Types of Data Structures

- **Tree:**



Types of Data Structures

- **Characteristics of a Tree:**
- The tree has various different characteristics which are as follows:
- A tree is also known as a Recursive data structure.
- In a tree, the Height of the root can be defined as the longest path from the root node to the leaf node.
- In a tree, one can also calculate the depth from the top to any node. The root node has a depth of 0.

Types of Data Structures

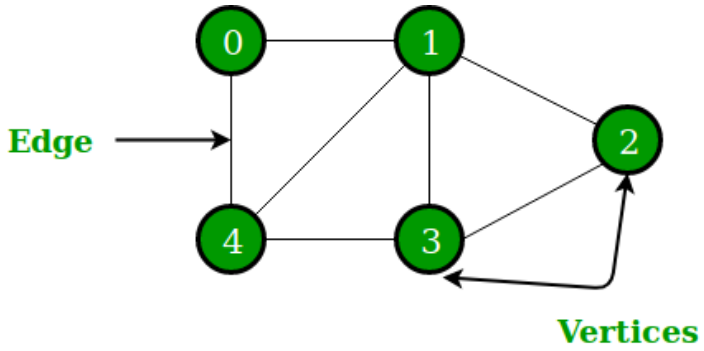
- **Operation performed on tree:**
- A tree is a non-linear data structure that consists of nodes connected by edges. Here are some common operations performed on trees:
- **Insertion:** New nodes can be added to the tree to create a new branch or to increase the height of the tree.
- **Deletion:** Nodes can be removed from the tree by updating the references of the parent node to remove the reference to the current node.
- **Search:** Elements can be searched for in a tree by starting from the root node and traversing the tree based on the value of the current node until the desired node is found.
- **Traversal:** The elements in a tree can be traversed in several different ways, including in-order, pre-order, and post-order traversal.
- **Height:** The height of the tree can be determined by counting the number of edges from the root node to the furthest leaf node.
- **Depth:** The depth of a node can be determined by counting the number of edges from the root node to the current node.

Types of Data Structures

- **Real-Life Applications of Tree:**
- File Systems.
- Internet and Web: The hierarchical structure of the Internet is represented by the Domain Name System (DNS) tree, which allows the mapping of domain names to IP addresses.
- A Decision Tree is an efficient machine-learning tool, commonly used in decision analysis. It has a flowchart-like structure that helps to understand data.
- Database Indexing: B-trees and other tree-based indexing structures are used in databases to efficiently retrieve and organize data for quick access.
- Game Development: In game development, trees are used for AI decision-making, such as behavior trees that control the actions of non-player characters (NPCs).

Types of Data Structures

- **Graph:**
- A graph is a non-linear data structure that consists of vertices (or nodes) and edges.
- It consists of a finite set of vertices and set of edges that connect a pair of nodes. The graph is used to solve the most challenging and complex programming problems.
- It has different terminologies which are Path, Degree, Adjacent vertices, Connected components, etc.



Types of Data Structures

- **Operation performed on Graph:**
- A graph is a non-linear data structure consisting of nodes and edges. Here are some common operations performed on graphs:
- **Add Vertex**: New vertices can be added to the graph to represent a new node.
- **Add Edge:** Edges can be added between vertices to represent a relationship between nodes.
- **Remove Vertex:** Vertices can be removed from the graph by updating the references of adjacent vertices to remove the reference to the current vertex.
- **Remove Edge:** Edges can be removed by updating the references of the adjacent vertices to remove the reference to the current edge.
- **Depth-First Search (DFS):** A graph can be traversed using a depth-first search by visiting the vertices in a depth-first manner.
- **Breadth-First Search (BFS):** A graph can be traversed using a breadth-first search by visiting the vertices in a breadth-first manner.

Types of Data Structures

- **Shortest Path**: The shortest path between two vertices can be determined using algorithms such as Dijkstra's algorithm.
- **Connected Components:** The connected components of a graph can be determined by finding sets of vertices that are connected to each other but not to any other vertices in the graph.
- **Cycle Detection:** Cycles in a graph can be detected by checking for back edges during a depth-first search.

Types of Data Structures

- **Real-Life Applications of Graph:**
- One of the most common real-world examples of a graph is Google Maps where cities are located as vertices and paths connecting those vertices are located as edges of the graph.
- A social network is also one real-world example of a graph where every person on the network is a node, and all of their friendships on the network are the edges of the graph.
- Computer Network: Graphs are employed in computer networks to model the connections between devices and routers, facilitating data routing and communication.

Thank You